iOS Forensics Cheat Sheet (iPhone/iPad)

Nikolaos Kranidiotis osec.gr

Quick Triage & Live Capture

- **Prevent Remote Wipe:** Immediately isolate the device from networks (enable Airplane Mode if unlocked, or use a Faraday bag). Remove any SIM cards to stop cellular data. This prevents remote wipe or tampering while evidence is collected.
- **Power State & Passcode:** Note if the device is powered on and if the screen is locked or unlocked. If unlocked, keep it awake (disable auto-lock in Settings) to maintain access. If locked and you cannot unlock, avoid powering off – a reboot will enable strong data protection (many

files become inaccessible until after next unlock).

- **Preserve Battery:** Keep the device charged during examination to prevent shutdown. A sudden power loss could lock data (device entering BFU - "Before First Unlock" state) and impede acquisition. Use a charging cable or portable battery if needed (avoid connecting to a computer unless necessary for acquisition).
- **Initial Documentation:** Photograph the device (screen on) at seizure. If the screen is unlocked, document any running apps or visible notifications with screenshots or photos. Note date/time displayed. This records ephemeral info (chat messages, timestamps) that might not be stored elsewhere.
- **Logical Backup:** If the device is accessible (unlocked or previously trusted), perform a quick backup using iTunes or `idevicebackup2` (with `--full` for a comprehensive backup). This captures user data (messages, calls, contacts, etc.) in a forensically sound manner. Do NOT set or change a backup password during this process (use an unencrypted backup unless one was already set).
- **Live Volatile Data:** If the device is unlocked or jailbroken, consider capturing a process list or network connections. For example, `ps -ax` or `top` (via SSH or terminal) can list running processes, and `netstat -an` can show open connections. iOS has no built-in shell, but a jailbreak enables these Linux commands. Any unusual running process or listening port could indicate malware. (Use caution: command execution on device may alter data – document any such actions.)

Key Data & Artifact Locations

- **Messages (SMS/iMessage):** Stored in SQLite DB `/private/var/mobile/Library/SMS/sms.db`. Contains SMS and iMessage texts, timestamps, participants. Attachments (images, etc.) are in `/private/var/mobile/Library/SMS/Attachments/` (organized in subfolders by GUID). Deleted messages may remain in the database until overwritten; iOS 16+ also retains recently deleted messages for 30 days in a separate folder (for recovery).
- **Call History:** Call logs (including cellular calls and FaceTime) are in `/private/var/mobile/Library/CallHistoryDB/CallHistory.storedata`. This is a SQLite database containing incoming/outgoing calls, timestamps, durations, and identifiers (phone numbers or contacts). Recent calls may sync across devices via iCloud, so this log can include calls made on other devices linked to the same Apple ID.
- **Contacts:** The address book is stored in `/private/var/mobile/Library/AddressBook/AddressBook-v22.abcddb` (and associated images in AddressBookImages.sqlitedb). It contains contact names, phone numbers, emails, etc. (On newer iOS, Contacts may also be synced to iCloud – check the Accounts database for iCloud contact sync status.)
- **Voicemail:** Visual voicemail messages are stored under `/private/var/mobile/Library/Voicemail/` (e.g. `voicemail.db` and audio files). The database holds metadata (caller, date, duration) and the folder contains the actual voicemail audio in AMR format (and any transcripts).
- **Notes:** Apple Notes are stored in a CoreData database. On iOS 10+, note contents reside in `/private/var/mobile/Containers/Shared/AppGroup/<GUID>/NoteStore.sqlite` (the GUID corresponds to the Notes app group). This database contains notes, their timestamps, and folder names.

Attached images are in the `NoteStore.sqlite` media folder. Older iOS versions used `/private/var/mobile/Library/Notes/` for note files.

- **Safari Browser:** Safari Internet history is in `/private/var/mobile/Library/Safari/History.db` (SQLite). Related Safari files include `LastSession.plist` and `BrowserState.db` (to track open tabs), and `Bookmarks.db` for bookmarks. Safari also maintains `CloudTabs.db` for iCloud-synced tabs across devices. Web page caches (including snapshots of visited pages) are stored under the app container in `.../Library/Caches/com.apple.Safari/`.
- **Mail:** Email accounts configured on the device are listed in `/private/var/mobile/Library/Accounts/Accounts4.sqlite`. Actual emails for the Apple Mail app are stored in `/private/var/mobile/Library/Mail/` (each account has a folder with an `Inbox`, `Sent`, etc.). Within those, emails are stored as EMLX files and an envelope index database tracks metadata. Attachments are typically in an "Attachments" subfolder for each account. (If using Gmail/IMAP, only recent or opened emails might be cached locally.)
- **Photos & Videos:** User photos and videos (Camera Roll) are in `/private/var/mobile/Media/DCIM/` (organized in 100APPLE, 101APPLE, etc. folders). Metadata for photos (including thumbnails, timestamps, and geolocation) is in `/private/var/mobile/Media/PhotoData/Photos.sqlite`. The Photos app also maintains databases for face recognition and other analysis. When photos/videos are deleted from the Photos app, they move to a "Recently Deleted" album (files remain in PhotoData or a .Trash folder for ~30 days).
- **Calendar & Reminders:** Calendar events are stored in `/private/var/mobile/Library/Calendar/Calendar.sqlitedb`, containing events, meetings, and alarms. Reminders are stored similarly in `/private/var/mobile/Library/Reminders/` (often a SQLite DB like `Reminders.sqlite` in an "Accounts" subdirectory). These include to-do items and their due dates/notes.
- **Location Data:** Significant Locations (frequently visited places used by Maps/Photos Memories) are stored in an encrypted database under `/private/var/mobile/Library/Caches/com.apple.routined/`. Wi-Fi network connections are logged in `/private/var/preferences/SystemConfiguration/com.apple.wifi.plist` (with SSIDs, last connection timestamps). Additional location breadcrumbs may be found in the consolidated SQLite databases (e.g. in `/private/var/locations/` on older iOS) and in the KnowledgeC database (which logs some location-related activities).

Persistence & Device Configuration

- **Launch Daemons:** iOS uses `launchd` for startup services. System launch daemons are in `/System/Library/LaunchDaemons/` and user agents (rarely present by default) in `/Library/LaunchAgents/`. On a stock device these should only contain Apple-signed items. If the device is jailbroken or compromised, check `/Library/LaunchDaemons/` (and `/Library/LaunchAgents/`) for any rogue plist that could launch malicious processes at boot.
- **Configuration Profiles:** Malicious or unauthorized `.mobileconfig` profiles can be installed to persist settings (VPN, Wi-Fi proxy, app whitelist, etc.). Examine `/private/var/mobile/Library/ConfigurationProfiles/` for any profiles. Also check `/private/var/containers/Shared/SystemGroup/systemgroup.com.apple.configurationprofiles/` for evidence of MDM or profiles. Profiles can grant an attacker control over VPN routing, certificate trust, or other security-relevant settings.
- **Provisioning Profiles:** Enterprise app provisioning profiles (which allow in-house or ad-hoc apps to run) are stored in `/private/var/`

`MobileDevice/ProvisioningProfiles/`. A malicious actor might install a fake enterprise-signed app; its provisioning profile would appear here. Look for any profiles that are suspicious or that the user would not likely have (especially on a personal device).

- **Property List Settings:** iOS doesn't have a registry; instead configuration is stored in plist files. System-wide settings live in `/private/var/preferences/` and user-level app settings in `/private/var/mobile/Library/Preferences/`. For example, `com.apple.springboard.plist` in the mobile Preferences contains SpringBoard settings (like if device auto-wipe on passcode failures is enabled). Examine relevant plists for evidence (e.g. the Wi-Fi plist for network info, trust store plists for installed certificates). The file `/private/var/root/Library/Lockdown/data_ark.plist` contains device identifying info (IMEI, phone number, etc.) which can also be useful to investigators.
- **Jailbreak Artifacts:** If the device was jailbroken (either by the user or malware), persistent modifications will exist. Common signs: presence of `Cydia.app` in /Applications, existence of `/private/var/lib/apt/` (Debian package lists), or additional Unix binaries (e.g. `ssh` daemon, `apt-get` command). A jailbreak often installs launch daemons (e.g. `substrated`) to re-enable itself on reboot. Identifying jailbreak files helps determine integrity (a jailbroken device may have had its security features bypassed, allowing malware to persist at root level).

Logs & Event History

- **Unified Logs:** iOS generates extensive unified logs (system and app logs) accessible via the `log` command or console. Log data is stored in compressed archives under `/private/var/db/diagnostics/` with corresponding metadata in `/private/var/db/uuidtext/`. These logs capture events like app launches, errors, network connections, etc. Unified logs require specialized parsing (e.g., using Apple's Console app on macOS or third-party tools) but can reveal timestamps for many actions (e.g., device unlock events, Wi-Fi connections, notifications). They can be huge; timestamp and keyword searches are essential (look for suspect app bundle IDs or error messages).
- **System Logs:** In addition to unified logs, iOS keeps specific logs for certain services. Examples: `MobileInstallation.log` (in `/private/var/installd/Library/Logs/MobileInstallation/`) records app install/uninstall activity, `MobileActivation.log` (in `/private/var/mobile/Library/Logs/mobileactivationd/`) logs device activation and boot sequences, and `Shutdown.log` (in `/private/var/db/diagnostics/`) notes the time of device shutdowns and any errors during power-off. These logs help establish timelines (e.g., when an app was installed or when the device was last rebooted).
- **Update & Restore Logs:** If the device was updated or restored, check `/private/var/mobile/MobileSoftwareUpdate/Restore.log`. This file logs firmware installs and can indicate if/when a device was wiped or iOS reinstalled. A recent restore could mean data on the device was reset at that time.
- **Crash Reports & Analytics:** iOS stores app crash reports under `/private/var/mobile/Library/Logs/CrashReporter/` (organized by app or by date). Analyze crash logs for clues of abnormal behavior (e.g., crashes of security-related processes). If "Analytics Data" is enabled (Settings > Privacy > Analytics), logs will appear in `Analytics-*.plist` or `.ips` files containing daily summaries and jetsam (memory pressure) events. These can show what processes were active and any unusual crashes (which might hint at exploitation attempts). Use `idevicecrashreport` to download all these logs easily.

- **KnowledgeC (Usage DB):** iOS's "Knowledge" database (CoreDuet) logs user activity events. Located at `/private/var/mobile/Library/CoreDuet/Knowledge/knowledgeC.db`, it contains records of app usage, device charging, screen on/off, location visits, etc. For example, it can show when a particular app was used and for how long, or when the user was moving vs stationary. This is extremely useful for building timelines (pattern-of-life analysis). Tools like iLEAPP or APOLLO can parse knowledgeC.db for human-readable output.

Deleted Data & File Recovery

- **APFS Snapshots:** iOS devices use the APFS file system, which can maintain snapshots (point-in-time copies of the file system). Snapshots may be created during iOS updates or by the system (and usually pruned quickly). If a snapshot exists in an image, mount it to recover files that were deleted or changed. (Use `tmutil listlocalsnapshots` on a mounted image in macOS, or forensic tools that support APFS snapshots.) Data in a snapshot could include files deleted from the live file system.
- **"Recently Deleted" Items:** Many iOS apps have a built-in recycle bin. Photos and Videos remain in the "Recently Deleted" album for ~30 days after deletion (files still on disk, marked for deletion). In Messages, as of iOS 16, deleted texts move to a "Recently Deleted" section (retained ~30 days). Forensically, check for these by using tools or by examining databases for flags that mark deletion timestamps. If the device or backup was acquired before that retention period lapsed, those items can be recovered.
- **SQLite Databases (Slack Space):** Key databases (SMS, Calls, Contacts, etc.) may contain deleted records in their free pages or Write-Ahead Log (WAL). Investigate the raw database with an SQLite forensic tool or script: you might retrieve messages or entries that the user deleted in-app but that remain in the database's unvacuumed space. For instance, run strings on the sms.db or use sqlparse tools to find remnants of old messages. Always work on a copy of the database.
- **File Carving:** With a full file system image (decrypted), standard file carving techniques can be applied to unallocated space. Use utilities like Foremost or PhotoRec to scan for common file signatures (pictures, documents). Due to file-based encryption, carving is only effective on areas of the image that were not encrypted or where encryption keys are available (on many modern iOS devices, most user files are encrypted at rest). Also consider carving the iTunes backup (manifest.plist will list files included; anything not present might have been deleted prior to backup).
- **Logs of Deletion:** Sometimes the act of deletion leaves a trace. For example, unified logs might have entries when the user deletes an app or when the system purges content (e.g., "JetsamEvent" logs if the OS purged an app for space). The KnowledgeC DB also records deletions (e.g., app uninstall events, file deletions as "filesystem events"). Timeline analysis can reveal if a bunch of data disappeared at a specific time, suggesting deletion.

Memory Acquisition & Analysis

- **Live RAM Capture:** There is no easy, supported way to image an iOS device's RAM. Unlike desktops, iPhones don't allow raw memory access. Attempts require exploits or jailbreaks. There are no native "dd" of /dev/mem on iOS. In practice, full memory dumps are rarely obtained, especially on newer devices with the Secure Enclave (which secures cryptographic material). Focus is usually on file system and keychain extraction rather than live RAM.
- **Checkm8 Exploit (Physical Access):** For iPhone 5s through iPhone X (A7–A11 chips), the unpatchable checkm8 bootrom exploit can be used to perform a low-level extraction. Tools (e.g., Elcomsoft iOS Forensic Toolkit, Cellebrite) use checkm8 to boot the device to a special ramdisk, enabling file system dumping and keychain decryption. This can retrieve data even if the device is locked (some file types remain encrypted until after first unlock, but a lot can be pulled). Note: checkm8 exploitation is read-only and considered forensically sound (does not alter user data).
- **Jailbreak-based Dumps:** If checkm8 is not available (newer devices), examiners might use a jailbreak (semi-tethered or otherwise) to access the file system. Jailbreaking a device (e.g., checkra1n, unc0ver for certain iOS versions) allows using SSH / SFTP to manually dump files, or run tools like `tar` to archive the data. However, jailbreaks modify the OS (leaving traces), so this method is less ideal. Use only if other methods fail, and document any changes.
- **Process Memory via Debugging:** In a research context (or jailbreak scenario), it's possible to attach a debugger to processes (using LLDB or Frida) to dump specific process memory (for example, grabbing an app's decrypted code or runtime data). This is advanced and not routine in forensic investigations, but can be useful for malware analysis when you need to extract code or data loaded in memory. Ensure this is done on a copy or after data extraction, as it can be invasive.
- **Volatile State via Sysdiagnose:** iOS has a feature called "sysdiagnose" (triggered by specific button presses or via profiles) which captures a snapshot of system state (including a limited memory snapshot, logs, running processes). If available, using sysdiagnose at time of collection can preserve some volatile information. The sysdiagnose archive (often found in `/private/var/tmp/` or retrieved via Xcode) contains diagnostic logs and some memory data (but not full RAM). This can be analyzed for insight into what was running at collection time.

Timeline Reconstruction

- **Timestamp Formats:** iOS stores timestamps in multiple formats. Common ones: Unix epoch (seconds since 1970, e.g. many SQLite "date" fields), Mac Absolute Time (seconds since Jan 1 2001, used in some Apple databases), and Cocoa timestamps (nanoseconds since 2001 in plists). Convert all timestamps to a uniform format (UTC) for comparison. For example, an SMS timestamp (Unix) and a KnowledgeC event (Mac Absolute) need conversion to correlate times. Watch out for timezone offsets (most timestamps are in UTC internally).
- **Cross-Artifact Correlation:** Build a timeline by combining artifacts: messages, calls, photos, app usage, location, etc. For example, a timeline might show: 10:30 AM - device unlock (from logs), 10:32 AM - SMS sent (from sms.db), 10:33 AM - photo taken (EXIF timestamp), 10:35 AM - email sent. By merging these, you can tell a story of what happened. Use tools like Excel or Timeline Explorer to sort and visualize events by time.

- **KnowledgeC & Usage Data:** The KnowledgeC database is a goldmine for timeline creation. It can show minute-by-minute usage (app opened, closed, device locked, location changes, etc.) with timestamps. Combining this with discrete events (like communications) fills in gaps. For instance, KnowledgeC might show the user was actively using a chat app around a certain time even if there's no message - useful to indicate activity.
- **File System Timestamps:** Don't overlook file MAC timestamps on the disk image. For instance, the filesystem "created" time of a video file in DCIM gives the moment a video was recorded. Likewise, if an app was installed or updated, the timestamps on its folder in `/Applications` or `/Containers` may reflect that. APFS stores file timestamps in nanosecond precision. Use `find` or SleuthKit to extract timestamps (e.g., `fls` to list files and create a body file, then `mactime` to make a timeline).
- **Tool Support:** Many mobile forensic suites have built-in timeline features (Cellebrite PA, Magnet AXIOM, etc.). They will automatically pull timestamps from various sources and let you filter by time. If doing manually, consider open-source tools like `maf.pl` (Make Apple Faster) or the APOLLO framework for parsing Apple "patterns of life" data into CSV timelines. Always verify critical timestamps with at least two sources (e.g., an SMS time vs the log entry for that SMS) to ensure accuracy.

Network & Cloud Artifacts

- **Wi-Fi Connections:** Known Wi-Fi network information is stored in `/private/var/preferences/SystemConfiguration/com.apple.wifi.plist`. This lists SSIDs the device has connected to, along with last connection timestamps and security type. Use this to place a device at certain locations (the presence of a home or work SSID, etc.). Additionally, `/private/var/db/dhcpclient/leases/` contains DHCP lease files that record IP addresses assigned to the iPhone on Wi-Fi networks, including date/time – useful for network timelines.
- **Bluetooth Devices:** Paired Bluetooth device info (names, MAC addresses, last seen) can be found in plists like `/private/var/mobile/Library/Preferences/com.apple.MobileBluetooth.devices.plist` and related databases. This can show if the phone was paired to a car, headphones, or other peripherals (which might indicate user habits or presence at certain places if the car kit connected).
- **VPN & Proxy Settings:** If a VPN was configured (manually or via profile), settings will be in `/private/var/preferences/SystemConfiguration/preferences.plist` under the "VPN" dictionary. Also check for any unusual proxy settings in that file or a `.mobileconfig` profile. A malicious profile might route traffic to an attacker-controlled server. Identifying any such network rerouting is crucial for detecting exfiltration.
- **iCloud Sync:** Determine what data was syncing to iCloud (which may mean some evidence is in the cloud, not on device). The Accounts database (Accounts4.sqlite) will list iCloud account and what services are enabled (iCloud Drive, Contacts, Messages in iCloud, etc.). If Messages in iCloud is on, older messages might not be on the device at all (only in cloud). Photos, if "Optimize Storage" is enabled, may have only thumbnails on device with full-resolution in iCloud. Note timestamps of last iCloud backup (found in `data_ark.plist` or in Settings info) – if an incident predates the last backup, the backup might contain the data even if device does not.
- **Third-Party Apps & Cloud:** Many apps store data server-side. WhatsApp, for instance, can have iCloud backups (if enabled) or Google Drive backups (Android) but on iOS the chat data is on device unless the user deletes it. Review any cloud storage apps (Dropbox, Google

Drive app, etc.) for signs of file uploads or transfers out of the device. Their app containers (under `/private/var/mobile/Containers/Data/Application/`) may have log files or cache indicating recent transfers.

- **Network Activity Artifacts:** iOS itself does not provide straightforward traffic logs, but some clues can be gathered: the "Analytics" logs sometimes list heavy data usage by apps; iOS devices also keep a cumulative data usage per app (in Settings, not easily accessible in files). If you suspect exfiltration, consider examining router logs or using network monitoring at time of seizure. On-device, unified logs may contain connection attempts or errors (search for keywords like "https" or specific hostnames in the log archives). Also, check for push notification traffic spikes or unusual persistent connections in the logs.

Malware Hunting & IOCs

- **Suspicious Apps:** Inventory installed apps. On a non-jailbroken iPhone, all apps should either be Apple default apps or from the App Store. If you find an app with no App Store metadata or a random bundle ID, investigate it. Use `ideviceinstaller -l` (or check the `Applications` or `Containers` directories) to list installed apps. A malicious app might use a generic icon or name to hide. Also check Settings > General > Profiles & Device Management for any enterprise apps that aren't expected.
- **Log Indicators:** Search the unified logs and analytics for known malicious patterns. For example, Pegasus spyware attacks were often detected by unusual crash logs and certain process names in diagnostics. Use IoC lists from threat intel (domains, file names) to grep through backups or file system dumps. Amnesty International's *Mobile Verification Toolkit (MVT)* is an open-source tool that scans iOS backups for signs of compromise (e.g., Pegasus domains in Safari history or iMessage). Running MVT or similar scripts on an iTunes backup of the device can highlight potential compromise.
- **Jailbreak/Exploit Traces:** Even if a malicious app was removed, traces of a jailbreak or exploit may remain. Check for leftover Cydia files, or odd binary names in `/usr/bin` (jailbreak tools often drop utilities there). Also, review the crash logs for any exploit attempt indicators (e.g., sandbox violations, Jetsam events around messaging apps could indicate exploit attempts). If the device has `Lockdown Mode` (iOS 16+) enabled, that might block some malware – note whether it was on or off during the period in question.
- **Unauthorized Access:** Look at the phone's authentication logs (if any). While iOS doesn't have traditional auth logs, the `Lockdown` records and Apple ID login activity may be visible. For example, `data_ark.plist` shows last unlock time and passcode attempt count (if available). If an attacker tried many passcodes, the device might show a back-off or "iPhone disabled" state (which would be evident). Additionally, check iCloud account login notifications (the user might have emails of logins from new locations).
- **Security App Data:** If the user had any security apps (e.g., antivirus, device management) installed, review their logs. While rare on iOS, there are MDM and monitoring apps that could either indicate the presence of an IT admin or, conversely, be spyware themselves (some spyware masquerades as device management). The presence of an MDM profile or app could either be legitimate (e.g., a corporate device) or malicious if unexpected.

Credential & Sensitive Data

- **Keychain (Passwords):** iOS devices securely store credentials in the Keychain database `/private/var/Keychains/keychain-2.db`. This includes Wi-Fi passwords, email passwords, website/app passwords (if using iCloud Keychain), and authentication tokens. The keychain is encrypted (tied to the device passcode and hardware keys). Forensic tools that perform full file system extraction will attempt to decrypt the keychain (requires either the passcode or exploiting keybag keys). Keychain items have access control classes; many are only accessible after the first device unlock.
- **Backup Keychain vs Device Keychain:** An unencrypted iTunes backup will not include most keychain items (for security). An encrypted backup does include the keychain, but that backup itself requires a password to open. If you have an encrypted backup and know/crack the password, you can retrieve keychain items from it. Otherwise, you need to extract on-device (with a jailbreak or exploit) to get the keychain. Make sure to note if a backup was encrypted (check `Manifest.plist` "IsEncrypted" field or if iTunes prompts for a password).
- **Keyboard Cache (Typing History):** iOS's keyboard learns from user input. This learned vocabulary is stored in `/private/var/mobile/Library/Keyboard/` (like `dynamic-text.dat`). It can contain things the user typed, including possible email addresses, search queries, and other phrases. In some cases, users may inadvertently type passwords into non-password fields, potentially storing them here. Review this file for any sensitive terms (it's a binary plist that can be parsed).
- **Apple Wallet & Payment Info:** Apple Wallet (passes and Apple Pay) stores data in `/private/var/mobile/Library/Passes/`, including payment cards (only device-specific tokens, not full card numbers) and boarding passes, etc. The file `passes23.sqlite` contains pass info (like last 4 digits of cards, transaction timestamps). Full credit card numbers or Apple Pay tokens are not accessible (secured by Secure Enclave). Also, any saved passwords or credit cards in Safari AutoFill are stored in the Keychain, not in plaintext.
- **Health & Sensitive Data:** Health app data (workouts, vital signs, etc.) is stored in `/private/var/mobile/Library/Health/healthdb_secure.sqlite` and is encrypted (requires the device passcode to decrypt). Health data and passwords are considered highly sensitive – if present, handle according to privacy requirements. Similarly, sensitive photos or notes might be protected with app-specific locks (Notes app allows locking notes with a password; those remain encrypted in the Notes database until unlocked by the user during normal use).
- **Personal Identifiers:** Device identifiers like serial number, IMEI, ICCID (SIM), and phone number can be found in `/private/var/root/Library/Lockdown/data_ark.plist` and through `ideviceinfo`. These link the device to accounts and subscriptions. Also check `Accounts4.sqlite` for usernames of accounts (email addresses, Apple ID). These pieces of info might be needed for legal process (e.g., to request iCloud data associated with an Apple ID or to identify a SIM's subscriber info).

Hidden & Temp Files

- **App Snapshots:** When apps are backgrounded, iOS takes a screenshot (for the app switcher). These images are stored per app in `Library/Caches/Snapshots/<bundle_id>/`. They can contain sensitive information from the app's last state (e.g., a banking app screen). Examine snapshot images for sensitive data that might not be elsewhere (they persist until the app is reopened and may then be updated or removed).
- **Application Caches:** App caches often contain traces of usage. For instance, Safari's cache (under its container in Caches/) may hold images from websites or search terms. Map apps cache map tiles and search addresses. Even if an app allows "private" mode, some data might still be cached. Look under `/private/var/mobile/Containers/Data/Application/<GUID>/Library/Caches/` for large or interesting files. Safari stores caches and even snapshots of visited pages (thumbnails) which could reveal browsing activity beyond the history DB.
- **Temporary Directories:** iOS has temp directories like `/private/var/tmp/` and `/private/var/mobile/Library/Caches/` that are not backed up and used for ephemeral data. Malware or exploits might drop files here. Check for any unusual files or scripts in these locations. Also look at `/private/var/mobile/Library/Logs/` for any temporary logs or crash logs that haven't been moved to the CrashReporter directory.
- **Orphaned Files:** If an app was uninstalled, its sandbox directory is usually deleted. However, sometimes files remain (e.g., in `/tmp` or in shared container folders). Additionally, remnants like notification attachments (in `/private/var/mobile/Library/PushStore`) or cached data in `/private/var/mobile/Library/Cache` might survive. Search the file system for keywords or known filenames from an app to see if anything lingers.
- **Device Logs:** Aside from app logs, iOS keeps low-level logs like baseband logs (cellular modem logs) which are not easily accessible without special tools, and possibly Apple Pay transaction logs. These typically require Apple internal tools to read, but be aware they exist. For instance, baseband logs (if enabled) might reside under `/private/var/logs/Baseband/` and could show cell tower connections. Access is limited, so these are usually only available via jailbreak or Apple diagnostics.

Forensic Tools & Techniques

- **Open-Source Utilities:** The `libimobiledevice` suite is invaluable for iOS forensics without official tools. Use `ideviceinfo` to get device info, `idevicebackup2` for backups, `idevicecrashreport` for logs, `idevicescreenshot` to capture the screen, and `ifuse` (on Linux/macOS) to mount the device's file system (if jailbroken or using AFC2). These can be used on any computer with the device's trust and do not require Apple's iTunes. Additionally, tools like *Mobile Verification Toolkit (MVT)* can scan backups for compromises, and *APOLLO/iLEAPP* can parse many iOS artifact databases for analysis.
- **Commercial Tools:** Numerous commercial forensic tools support iOS extraction and analysis. Leading ones include Cellebrite UFED/Physical Analyzer, GrayKey (by Grayshift) for unlocking devices via passcode brute-force or exploit, Magnet AXIOM, Oxygen Forensic Detective, and Elcomsoft iOS Forensic Toolkit. These tools can perform full file system extractions (often using exploits like checkm8 or proprietary methods), decode hundreds of artifact types, and help bypass lock screens on supported devices. They also maintain chain-of-custody and generate reports suitable for court. Always use the latest version due to frequent iOS updates closing vulnerabilities.

- **Device Handling:** Remember standard practices: keep device Faraday isolated until you need to enable communications (e.g., pairing with a forensic workstation), document every step (photographs, notes), and whenever possible, perform your work on a copy of the data (e.g., an extracted disk image or backup) rather than the live device. iOS extractions can be complex—when in doubt, consult vendor documentation or the community (e.g., forensic focus forums) for the latest methods specific to your device/iOS version.